



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/617,592	07/11/2003	David C. Kung	124263-1013	3712
<div>Gardere Wynne Sewell LLP 3000 Thanksgiving Tower Suite 3000 1601 Elm Street Dallas, TX 75201-4767</div>				
			<div>EXAMINER YIGDALL, MICHAEL J</div>	
			<div>ART UNIT 2192</div>	<div>PAPER NUMBER</div>
			<div>MAIL DATE 10/16/2007</div>	<div>DELIVERY MODE PAPER</div>

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

## Office Action Summary

Application No.

10/617,592

Applicant(s)

KUNG ET AL.

Examiner

Michael J. Yigdoll

Art Unit

2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

### Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

### Status

- 1) ☒ Responsive to communication(s) filed on 18 July 2007.
- 2a) ☒ This action is FINAL. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

### Disposition of Claims

- 4) ☒ Claim(s) 1-28 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-28 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

### Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on \_\_\_\_\_ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

### Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

### Attachment(s)

- |                                                                                                            |                                                                                         |
|------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| 1) <input type="checkbox"/> Notice of References Cited (PTO-892)                                           | 4) <input type="checkbox"/> Interview Summary (PTO-413)<br>Paper No(s)/Mail Date. _____ |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948)                       | 5) <input type="checkbox"/> Notice of Informal Patent Application                       |
| 3) <input type="checkbox"/> Information Disclosure Statement(s) (PTO/SB/08)<br>Paper No(s)/Mail Date _____ | 6) <input type="checkbox"/> Other: _____                                                |

**DETAILED ACTION**

1. This Office action is responsive to Applicant's submission filed on July 18, 2007. Claims 1-28 are pending.

***Response to Amendment***

2. The objection to the specification has been withdrawn in view of Applicant's amendments.
3. The objection to claim 8 has been withdrawn in view of Applicant's amendments.
4. The rejection of claim 10 under 35 U.S.C. 112, second paragraph, has been withdrawn in view of Applicant's amendments.
5. The rejection of claims 16-28 under 35 U.S.C. 101 has been withdrawn in view of Applicant's amendments.

***Response to Arguments***

6. Applicant's arguments have been fully considered but they are not persuasive.

Applicant contends that Coad does not teach or suggest extracting at least one single method call from each of the at least one complex method call where the extracting includes recursively replacing the at least one single method call with a phase variable (remarks, page 16).

However, as set forth in the Office action, Pennell teaches sensing a complex method call with which a plurality of methods is associated (see, for example, column 2, line 65 to column 3, line 5), and extracting a number of single method calls from such a complex method call (see, for

Art Unit: 2192

example, column 5, lines 29-40). Broughton teaches recursively substituting a phase variable for each component of a complex method call (see, for example, paragraphs [0068]-[0070]). Thus, the combination of references teaches or suggests extracting at least one single method call from each of the at least one complex method call where the extracting includes recursively replacing the at least one single method call with a phase variable. One cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981), and *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Furthermore, while Applicant notes that Coad teaches checking for multiple assignments but contends that Coad does not teach or suggest replacing these multiple assignments with a phase variable (remarks, page 16), the examiner respectfully submits that, as set forth in the Office action, Coad does suggest recursively substituting a phase variable for each component of a complex assignment statement. For example, Coad illustrates breaking the complex assignment statement “ $i = (j = 25) + 30$ ” into two components “ $j = 25$ ” and “ $i = j + 30$ ,” where the variables “ $i$ ” and “ $j$ ” are reasonably considered phase variables (see FIGS. 8B and 8C).

Applicant contends that Pennell does not teach or suggest examining source code, and further contends that Pennell does not disclose replacing components of a complex method call or recursively replacing at least one single method call included in a complex method call with a phase variable (remarks, page 16).

However, Coad teaches examining source code (see, for example, column 6, lines 18-22), and Broughton teaches recursively substituting a phase variable for each component of a complex method call (see, for example, paragraphs [0068]-[0070]). Again, one cannot show

Art Unit: 2192

nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981), and *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Applicant contends that Broughton and the OMG reference do not teach or suggest recursively replacing at least one single method call included in a complex method call with a phase variable (remarks, page 16).

However, the examiner respectfully submits that Broughton does teach recursively substituting a phase variable for each component of a complex method call (see, for example, paragraphs [0068]-[0070]). Here, as set forth in the Office action, Broughton teaches substituting a phase variable “temp = a & b & c” for the component “a & b & c” of the complex method call “decrement (d, a & b & c).” Pennell further teaches an analogous complex method call “total (get first value, get second value)” that includes at least one single method call such as “get first value” (see, for example, column 2, line 65 to column 3, line 5). Thus, the combination of references teaches or suggests recursively replacing at least one single method call included in a complex method call with a phase variable. Again, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981), and *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Applicant is respectfully reminded that the test for obviousness is not that the claimed invention must be expressly suggested in any one or all of the references. Rather, the test is what the combined teachings of the references would have suggested to those of ordinary skill in the

Art Unit: 2192

art. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981). The examiner respectfully submits that the combined teachings of the references would have suggested the claimed subject matter to those of ordinary skill in the art.

***Claim Rejections - 35 USC § 112***

7. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

8. Claim 2 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter that Applicant regards as the invention.

With respect to claim 2 (currently amended), the claim recites “a method parameter of the at least one method call.” However, there is insufficient antecedent basis for “the at least one method call” in the claims. Specifically, claim 2 is indefinite as to whether “the at least one method call” refers to the “at least one complex method call” or the “at least one single method call” recited in claim 1. The examiner’s interpretation is that “the at least one method call” is intended to refer to the “at least one complex method call.”

***Claim Rejections - 35 USC § 103***

9. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

10. Claims 1-9 and 11-28 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,851,107 to Coad et al. (art of record, "Coad") in view of U.S. Patent No. 6,598,181 to Pennell (art of record, "Pennell") and in view of U.S. Pub. No. 2003/0188299 to Broughton et al. (art of record, "Broughton").

With respect to claim 1 (currently amended), Coad discloses a process for providing a representation of specified characteristics of a previously developed object-oriented software program (see, for example, column 4, lines 38-41, which shows providing a representation of a program, and column 15, line 61 to column 16, line 4, which further shows that the program was previously developed in an object-oriented language), the software program including a plurality of object classes and further including object related methods belonging to respective object classes (see, for example, column 5, lines 14-20, which shows that the program includes object classes and methods).

Coad further discloses sensing methods and measuring the complexity of the object classes by examining the source code of the software program (see, for example, column 6, lines 18-22, and column 7, Table 3), but does not expressly disclose said process comprising:

sensing at least one complex method call by examining the source code of the software program, a plurality of the methods being associated with each of the at least one complex method call, wherein the at least one complex method call includes at least one single method call; and

extracting the at least one single method call from each of the at least one complex method call, the extracting comprises recursively replacing the at least one single method call with a phase variable.

However, in an analogous art, Pennell discloses sensing at least one complex method call included in a software program, a plurality of methods being associated with the complex method call (see, for example, column 2, line 65 to column 3, line 5, which shows sensing a complex method call such as “total (get first value, get second value)” that includes single method calls such as “get first value” and “get second value”). Pennell further discloses extracting a number of single method calls from the complex method call (see, for example, column 5, lines 29-40).

Coad is directed to helping a programmer understand and visualize a program (see, for example, column 1, lines 38-46). Likewise, Pennell discloses that extracting single method calls from complex method calls helps a programmer understand and debug a program (see, for example, column 1, line 21-33 and column 2, lines 41-50).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include in Coad, sensing at least one complex method call by examining the source code of the software program, a plurality of the methods being associated with each of the at least one complex method call, wherein the at least one complex method call includes at least one single method call, and extracting the at least one single method call from each of the at least one complex method call, so as to further improve program understanding.

Coad in view of Pennell does not expressly disclose that the extracting comprises recursively replacing the at least one single method call with a phase variable.



Nonetheless, Coad suggests recursively replacing a component of a complex assignment statement with a phase variable (see, for example, FIGS. 8B and 8C).

Furthermore, in an analogous art, Broughton discloses recursively replacing a component of a complex method call with a phase variable (see, for example, paragraphs [0068]-[0070], which shows replacing a component “a & b & c” of a complex method call “decrement (d, a & b & c)” with a phase variable “temp = a & b & c”). Broughton discloses that such an expansion or extraction step facilitates later binding and aliasing optimizations (see, for example, paragraph [0071]).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Coad and Pennell such that the extracting comprises recursively replacing the at least one single method call with a phase variable, as Coad suggests, so as to facilitate later binding and aliasing optimizations, as Broughton teaches.

Coad in view of Pennell and Broughton further discloses the process comprising:

generating a set of information for each of the methods from the at least one single method call, the information set for a particular method containing at least a name of the particular method and the object class to which the particular method belongs (see, for example, Coad, column 4, lines 47-51, which shows generating a model of the program, and column 5, lines 4-20, which further shows that the model includes information for each method and the class to which it belongs, such as in the example illustrated in FIGS. 4 and 5); and

constructing a representation of interactions between objects of the software program from the information contained in said method information sets (see, for example, Coad, column 4, lines 52-54, which shows constructing a representation of the program from the model, and

Art Unit: 2192

column 17, lines 2-8, which further shows that the representation depicts interactions among objects of the program).

With respect to claim 2 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the at least one single method call comprises a method parameter of the at least one method call or a continuous method call (see, for example, Pennell, column 2, line 65 to column 3, line 5, which shows that the single method call “get first value” is a method parameter of the complex method call “total (get first value, get second value)”).

With respect to claim 3 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the process further comprises extracting the name and class of each of the methods from the software program (see, for example, Coad, column 5, lines 4-20, which shows extracting the name and class of each method to include in the model, such as in the example illustrated in FIGS. 4 and 5).

With respect to claim 4 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the extracting comprises output a given complex method call as multiple lines, wherein at least one of the multiple lines contains the at least one single method call (see, for example, Coad, FIGS. 8B and 8C, which shows that the complex assignment statement is output as multiple lines, and Broughton, paragraphs [0068]-[0070], which further shows that the complex method call is output as multiple lines).

With respect to claim 5 (currently amended), the rejection of claim 4 is incorporated, and Coad in view of Pennell and Broughton further discloses that the extracting comprises:

a first parsing phase configured to separate any casting operations included in the given complex method call (see, for example, Coad, column 14, lines 43-44, which shows separating any casting operations, such as casting operations that are considered unnecessary);

a second parsing phase configured to isolate any method parameters included in the given complex method call (see, for example, Pennell, column 2, line 65 to column 3, line 5, which shows isolating any method parameters, such as the method parameter "get first value"); and

a third parsing phase configured to resolve any continuous method calls included in the given complex method call into multiple lines, each containing one of the at least one single method call (see, for example, Pennell, column 5, lines 1-40, which shows resolving any continuous method calls into multiple lines, such as the multiple "call" instructions).

With respect to claim 6 (currently amended), the rejection of claim 5 is incorporated, and Coad in view of Pennell and Broughton further discloses that the first parsing phase is implemented prior to the second parsing phase, and the second parsing phase is implemented prior to said third parsing phase (see, for example, Coad, FIG. 9, which shows that the parsing phases are implemented in sequence at step 906).

With respect to claim 7 (currently amended), the rejection of claim 6 is incorporated, and Coad in view of Pennell and Broughton further discloses that the generating includes parsing an output provided by the third parsing phase to determine the correct object class for each of the methods (see, for example, Coad, column 16, lines 13-16, which shows parsing the program to

Art Unit: 2192

generate the model, and column 5, lines 4-20, which shows that the model identifies the class of each method, such as in the example illustrated in FIGS. 4 and 5).

With respect to claim 8 (currently amended), the rejection of claim 7 is incorporated, and Coad in view of Pennell and Broughton further discloses that the process further comprises determining whether a method is a user-defined method or a standard application programming interface method (see, for example, Coad, column 14, lines 25-28 and 37-43, which shows determining whether a method is a method defined in the source code or an imported method, and column 14, lines 29-33, which further shows identifying standard API methods imported from the "java.lang" package).

With respect to claim 9 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the constructing comprises constructing a sequence diagram depicting the interactions between respective objects of the software program (see, for example, Coad, FIG. 14 and column 17, lines 2-8, which shows constructing a sequence diagram that depicts the interactions among objects of the program).

With respect to claim 11 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the software program is in the form of source code (see, for example, Coad, column 15, lines 61-64, which shows that the program is in the form of source code).

With respect to claim 12 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the software program is written

in JAVA™ software code (see, for example, Coad, column 16, lines 1-4, which shows that the program is written in the Java language).

With respect to claim 13 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that the software program is written in C++ software code (see, for example, Coad, column 16, lines 1-4, which shows that the program is written in the C++ language).

With respect to claim 14 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that at least one of the object related methods in the software program is a polymorphic method (see, for example, Coad, column 9, Table 8, which shows the polymorphism of methods in the program).

With respect to claim 15 (currently amended), the rejection of claim 1 is incorporated, and Coad in view of Pennell and Broughton further discloses that at least one of the object related methods in the software program is related to an inheritance feature, and the extraction includes tracking an inheritance path until it reaches a parent object class to which the method is defined (see, for example, Coad, column 8, Table 6, which shows tracking an inheritance tree to identify where a class and its methods are defined).

With respect to claim 16 (currently amended), Coad discloses a computer program product in a computer readable media for providing a representation of specified characteristics of a previously developed object-oriented software program (see, for example, column 4, lines 38-41, which shows providing a representation of a program, and column 15, line 61 to column

Art Unit: 2192

16, line 4, which further shows that the program was previously developed in an object-oriented language), the software program including a plurality of object classes and object related single methods belonging to respective object classes (see, for example, column 5, lines 14-20, which shows that the program includes object classes and methods).

Coad further discloses that the program includes method calls and further discloses measuring the complexity of the object classes by examining the source code of the software program (see, for example, column 6, lines 18-22, and column 7, Table 3), but does not expressly disclose the software program further including at least one complex method call containing a plurality of the single methods, the computer program product comprising:

first instructions for extracting a plurality of individual method calls from each of the at least one complex method call.

However, in an analogous art, Pennell discloses a software program that includes at least one complex method call containing a plurality of single methods (see, for example, column 2, line 65 to column 3, line 5, which shows a complex method call such as “total (get first value, get second value)” that includes single method calls such as “get first value” and “get second value”). Pennell further discloses extracting a number of individual method calls from the complex method call (see, for example, column 5, lines 29-40).

Coad is directed to helping a programmer understand and visualize a program (see, for example, column 1, lines 38-46). Likewise, Pennell discloses that extracting individual method calls from complex method calls helps a programmer understand and debug a program (see, for example, column 1, line 21-33 and column 2, lines 41-50).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include in Coad, first instructions for extracting a plurality of individual method calls from each of the at least one complex method call, so as to further improve program understanding.

Coad in view of Pennell does not expressly disclose that the first instructions comprise recursively replacing each of the single methods with a phase variable.

Nonetheless, Coad suggests recursively replacing a component of a complex assignment statement with a phase variable (see, for example, FIGS. 8B and 8C).

Furthermore, in an analogous art, Broughton discloses recursively replacing a component of a complex method call with a phase variable (see, for example, paragraphs [0068]-[0070], which shows replacing a component “a & b & c” of a complex method call “decrement (d, a & b & c)” with a phase variable “temp = a & b & c”). Broughton discloses that such an expansion or extraction step facilitates later binding and aliasing optimizations (see, for example, paragraph [0071]).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Coad and Pennell such that the first instructions comprise recursively replacing each of the single methods with a phase variable, as Coad suggests, so as to facilitate later binding and aliasing optimizations, as Broughton teaches.

Coad in view of Pennell and Broughton further discloses the computer program product comprising:

second instructions for storing in a data base a set of information for each of the single methods, the set of information for a particular single method containing at least a name of the

Art Unit: 2192

particular method and the object class to which the particular method belongs (see, for example, Coad, column 4, lines 47-51, which shows storing a model of the program, and column 5, lines 4-20, which further shows that the model includes information for each method and the class to which it belongs, such as in the example illustrated in FIGS. 4 and 5); and

third instructions for graphically constructing and graphically outputting a representation of interactions between objects of the software program from the information contained in the set of information (see, for example, Coad, column 4, lines 52-54, which shows constructing a representation of the program from the model, and column 17, lines 2-8, which further shows that the representation depicts interactions among objects of the program).

With respect to claim 17 (currently amended), the rejection of claim 16 is incorporated, and Coad in view of Pennell and Broughton further discloses fourth instructions for extracting the name and the object class of each of the single methods from the software program (see, for example, Coad, column 5, lines 4-20, which shows extracting the name and class of each method to include in the model, such as in the example illustrated in FIGS. 4 and 5).

With respect to claim 18 (currently amended), the rejection of claim 17 is incorporated, and Coad in view of Pennell and Broughton further discloses that the first instructions comprise fifth instructions for outputting a given complex method call as multiple lines, each containing one of the single method calls (see, for example, Coad, FIGS. 8B and 8C, which shows that the complex assignment statement is output as multiple lines, and Broughton, paragraphs [0068]-[0070], which further shows that that the complex method call is output as multiple lines).



With respect to claim 19 (currently amended), the rejection of claim 18 is incorporated, and Coad in view of Pennell and Broughton further discloses that the first instructions sequentially implement a first parsing phase to separate any casting operations included in the given complex method call (see, for example, Coad, column 14, lines 43-44, which shows separating any casting operations, such as casting operations that are considered unnecessary), a second parsing phase to isolate any method parameters included in the given complex method call (see, for example, Pennell, column 2, line 65 to column 3, line 5, which shows isolating any method parameters, such as the method parameter “get first value”), and a third parsing phase to resolve the given complex method call into multiple lines, each containing one of the single method calls (see, for example, Pennell, column 5, lines 1-40, which shows resolving any continuous method calls into multiple lines, such as the multiple “call” instructions).

With respect to claim 20 (currently amended), the rejection of claim 19 is incorporated, and Coad in view of Pennell and Broughton further discloses the third instructions constructs a sequence diagram depicting the interactions between respective objects of the software program (see, for example, Coad, FIG. 14 and column 17, lines 2-8, which shows constructing a sequence diagram that depicts the interactions among objects of the program).

With respect to claim 21 (currently amended), the rejection of claim 20 is incorporated, and Coad in view of Pennell and Broughton further discloses that the software program is in the form of source code (see, for example, Coad, column 15, lines 61-64, which shows that the program is in the form of source code).

With respect to claim 22 (currently amended), Coad discloses an apparatus for providing a sequence diagram representing specified characteristics of a previously developed object-oriented software program (see, for example, column 4, lines 38-41, which shows providing a representation of a program, and column 15, line 61 to column 16, line 4, which further shows that the program was previously developed in an object-oriented language, and see, for example, FIG. 14 and column 17, lines 2-8, which shows that the representation is a sequence diagram), said program including a plurality of object classes and further including object related methods belonging to respective object classes (see, for example, column 5, lines 14-20, which shows that the program includes object classes and methods), the apparatus comprising:

a computer system having a display (see, for example, FIG. 6).

Coad further discloses sensing methods and measuring the complexity of the object classes by examining the source code of the software program (see, for example, column 6, lines 18-22, and column 7, Table 3), but does not expressly disclose the apparatus comprising:

means for sensing at least one complex method call by examining the source code of the software program, a plurality of the methods being associated with each of the at least one complex method call;

Method Detail Parser means for extracting a plurality of single method calls from each of the at least one complex method call.

However, in an analogous art, Pennell discloses sensing at least one complex method call included in a software program, a plurality of methods being associated with the complex method call (see, for example, column 2, line 65 to column 3, line 5, which shows sensing a complex method call such as “total (get first value, get second value)” that includes single

Art Unit: 2192

method calls such as “get first value” and “get second value”). Pennell further discloses extracting a number of single method calls from the complex method call (see, for example, column 5, lines 29-40).

Coad is directed to helping a programmer understand and visualize a program (see, for example, column 1, lines 38-46). Likewise, Pennell discloses that extracting single method calls from complex method calls helps a programmer understand and debug a program (see, for example, column 1, line 21-33 and column 2, lines 41-50).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include in Coad, means for sensing at least one complex method call by examining the source code of the software program, a plurality of the methods being associated with each of the at least one complex method call, and Method Detail Parser means for extracting a plurality of single method calls from each of the at least one complex method call, so as to further improve program understanding.

Coad in view of Pennell does not expressly disclose that the Method Detail Parser means comprises recursively replacing the associated methods with a phase variable.

Nonetheless, Coad suggests recursively replacing a component of a complex assignment statement with a phase variable (see, for example, FIGS. 8B and 8C).

Furthermore, in an analogous art, Broughton discloses recursively replacing a component of a complex method call with a phase variable (see, for example, paragraphs [0068]-[0070], which shows replacing a component “a & b & c” of a complex method call “decrement (d, a & b & c)” with a phase variable “temp = a & b & c”). Broughton discloses that such an expansion or

Art Unit: 2192

extraction step facilitates later binding and aliasing optimizations (see, for example, paragraph [0071]).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Coad and Pennell such that the Method Detail Parser means comprises recursively replacing the associated methods with a phase variable, as Coad suggests, so as to facilitate later binding and aliasing optimizations, as Broughton teaches.

Coad in view of Pennell and Broughton further discloses the apparatus comprising:

means for generating a set of information for each of the object related methods from the single method calls, the set of information for a particular object related method containing at least a name of the particular method and the object class to which the particular method belongs (see, for example, Coad, column 4, lines 47-51, which shows generating a model of the program, and column 5, lines 4-20, which further shows that the model includes information for each method and the class to which it belongs, such as in the example illustrated in FIGS. 4 and 5); and

means for constructing a sequence diagram representing interactions between objects of the software program from the information contained in the sets of information and outputting the sequence diagram on the display (see, for example, Coad, column 4, lines 52-54, which shows constructing a representation of the program from the model, and FIG. 14 and column 17, lines 2-8, which further shows that the representation is a sequence diagram that depicts interactions among objects of the program).

With respect to claim 23 (currently amended), the rejection of claim 22 is incorporated, and Coad in view of Pennell and Broughton further discloses that the apparatus further comprises

Method Information Parser means for extracting the name and the object class of each of the methods from said software program (see, for example, Coad, column 5, lines 4-20, which shows extracting the name and class of each method to include in the model, such as in the example illustrated in FIGS. 4 and 5).

With respect to claim 24 (currently amended), the rejection of claim 23 is incorporated, and Coad in view of Pennell and Broughton further discloses that the Method Detail Parser means is operable to output a given complex method call as multiple lines, each containing one of the single method calls (see, for example, Coad, FIGS. 8B and 8C, which shows that the complex assignment statement is output as multiple lines, and Broughton, paragraphs [0068]-[0070], which further shows that that the complex method call is output as multiple lines).

With respect to claim 25 (currently amended), the rejection of claim 24 is incorporated, and Coad in view of Pennell and Broughton further discloses that the Method Detail Parser means is configured to separate any casting operations included in the given complex method call during a first parsing phase (see, for example, Coad, column 14, lines 43-44, which shows separating any casting operations, such as casting operations that are considered unnecessary), to isolate any method parameters included therein during a second parsing phase (see, for example, Pennell, column 2, line 65 to column 3, line 5, which shows isolating any method parameters, such as the method parameter “get first value”), and resolve the given complex method call into multiple lines, each containing one of the single method calls, during a third parsing phase (see, for example, Pennell, column 5, lines 1-40, which shows resolving any continuous method calls into multiple lines, such as the multiple “call” instructions).

With respect to claim 26 (currently amended), the rejection of claim 25 is incorporated, and Coad in view of Pennell and Broughton further discloses that the first parsing phase is implemented prior to the second parsing phase, and the second parsing phase is implemented prior to the third parsing phase (see, for example, Coad, FIG. 9, which shows that the parsing phases are implemented in sequence at step 906).

With respect to claim 27 (currently amended), the rejection of claim 26 is incorporated, and Coad in view of Pennell and Broughton further discloses that the constructing means comprises a drawing engine for depicting interactions between respective objects of the software program (see, for example, Coad, column 17, lines 2-8, which shows depicting interactions among objects of the program).

With respect to claim 28 (currently amended), the rejection of claim 27 is incorporated, and Coad in view of Pennell and Broughton further discloses that the software program is in the form of source code (see, for example, Coad, column 15, lines 61-64, which shows that the program is in the form of source code).

11. Claim 10 is rejected under 35 U.S.C. 103(a) as being unpatentable over Coad in view of Pennell and Broughton, as applied to claim 9 above, and further in view of the *OMG Unified Modeling Language Specification Version 1.3* (art of record, "OMG").

With respect to claim 10, the rejection of claim 9 is incorporated. Coad in view of Pennell and Broughton does not expressly disclose that the sequence diagram displays a

Art Unit: 2192

condition of a method call to indicate that the method call occurs only when the condition is evaluated to be true.

Nonetheless, Coad suggests that the sequence diagram conforms to the Unified Modeling Language (see, for example, column 15, lines 50-57). It is well known in the art that sequence diagrams conforming to the Unified Modeling Language (UML) display the condition of a method call to indicate that the call occurs only when the condition is evaluated to be true. For example, OMG illustrates a UML sequence diagram that displays the condition “[x > 0]” of a method call “foo(x)” to indicate that the call occurs only when the condition is evaluated to be true (see, for example, Figure 3-48 on page 3-97).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made that the sequence diagram of Coad displays the condition of a method call to indicate that the call occurs only when the condition is evaluated to be true.

### *Conclusion*

12. Applicant's amendment necessitated any new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire **THREE MONTHS** from the mailing date of this action. In the event a first reply is filed within **TWO MONTHS** of the mailing date of this final action and the advisory action is not mailed until after the end of the **THREE-MONTH** shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event,

Art Unit: 2192

however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

13. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Michael J. Yigdall whose telephone number is (571) 272-3707. The examiner can normally be reached on Monday through Friday from 7:30am to 4:00pm.

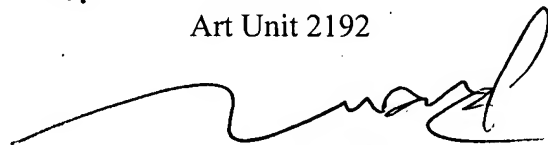
If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

My

Michael J. Yigdall  
Examiner  
Art Unit 2192

mjy

  
TUAN DAM  
SUPERVISORY PATENT EXAMINER